Frédérique Mayer, MdC
Frederique.Mayer@ensgsi.inpl-nancy.fr

Pôle Ingénierie des Systèmes

IS5 Base de Données

ENSGSI
LES INGÉNIEURS, MANAGERS DE L'INNOVATION

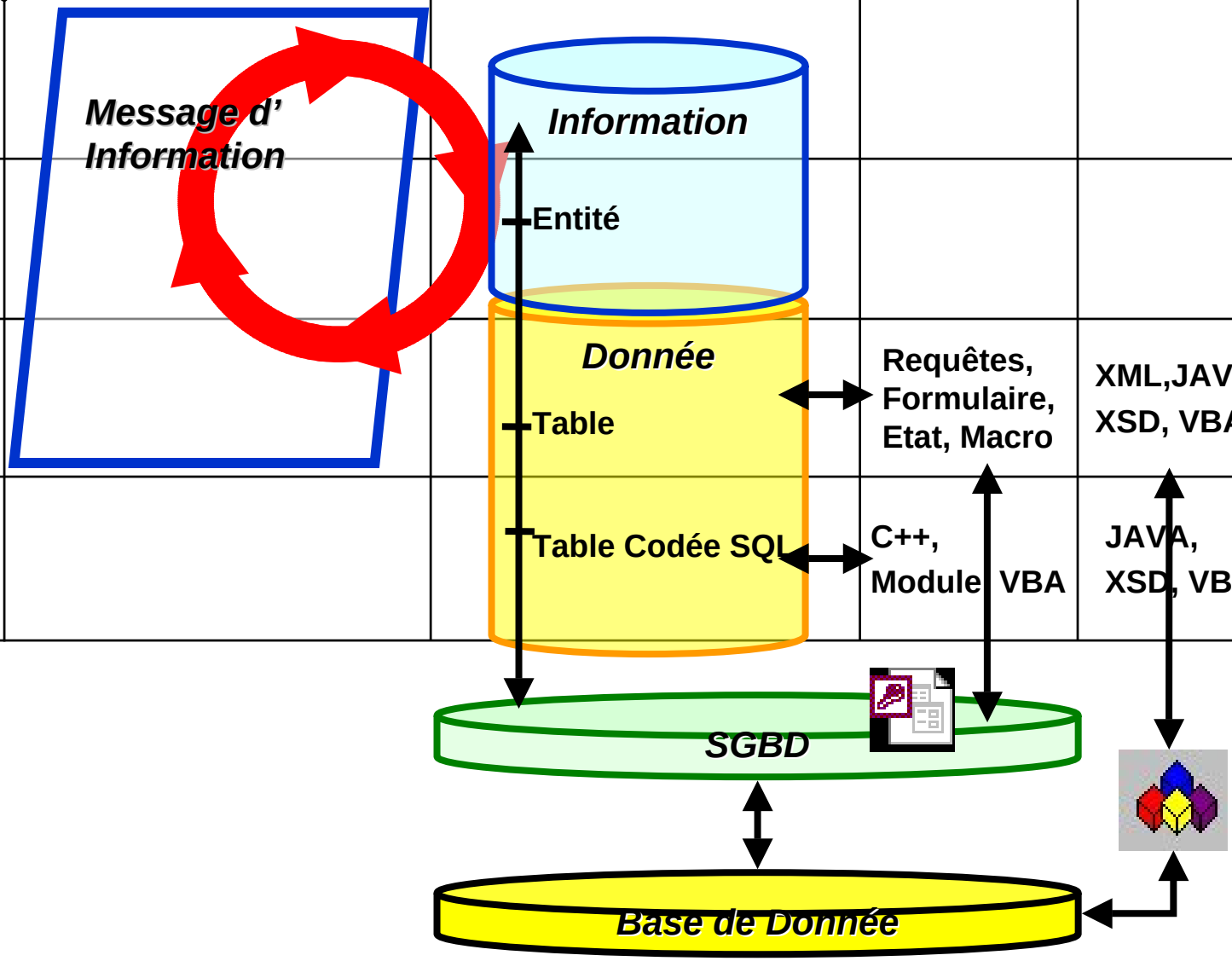
Modèles, Méthodes, Théories

pour concevoir une BASE DE DONNEES

- Approche systémique relationnelle MErise Gamma : modèles du système d'information

Modèle du Système / Cycle d'abstraction	Communication (E)	Mémorisation (T)	Traitement (F)	
			Transaction	Transformation
Conceptuel	MCC	MCD	MCT	CVO (RdP)
Organisationnel	MOC (MF)	MOD	MOT	CVO (G7)
Opérationnel	MLC (MAA)	MLO (MR)	MLTI	MLTNI (SFC)
Technique	MAT			

Modèle du Système / Cycle d'abstraction	Communication de l'information	Mémorisation de l'information	Traitement de l'information	
			Transac-tionnel	Transforma-tionnel
Conceptuel				
Organisationnel				
Opérationnel				
Technique				



B. Modèle Physique de Données

- Afin d'exprimer les différentes relations de données et entre données, il convient d'utiliser un langage supportant cet aspect relationnel de la Base de Données : SQL (Structured Query Language) normalisé.

- SQL permet de coder le modèle relationnel de données, c'est à dire de coder les différentes tables, associations, cardinalités ainsi que les requêtes permettant de manipuler les données

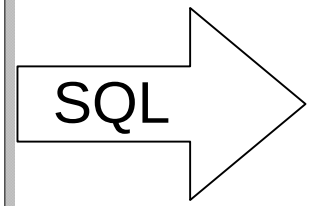
B. Modèle Physique de Données

- Le Langage SQL est composé d'une syntaxe, d'une sémantique, d'une grammaire. Il comporte quatre sous-langages :
  - un langage de définition de schémas pour décrire les données (relations et relations entre relations)
  - un langage de manipulation de données pour accéder à la base de données et pour manipuler les données
  - un langage pour réaliser une interface entre un SGBD et un programme d'application écrit dans un autre code
  - une syntaxe pour inclure des éléments de SQL dans un autre programme d'application écrit dans un autre code

1. SQL, un langage pour décrire des données

- Expression d'une table relationnelle à l'aide de SQL

Etudiant	
N_Securite_Sociale INTEGER	
N_Etudiant INTEGER	*Indx*
nomEtudiant CHAR(0)	*Indx*
prenom CHAR(0)	
ville CHAR(0)	
cursus CHAR(0)	
region CHAR(0)	

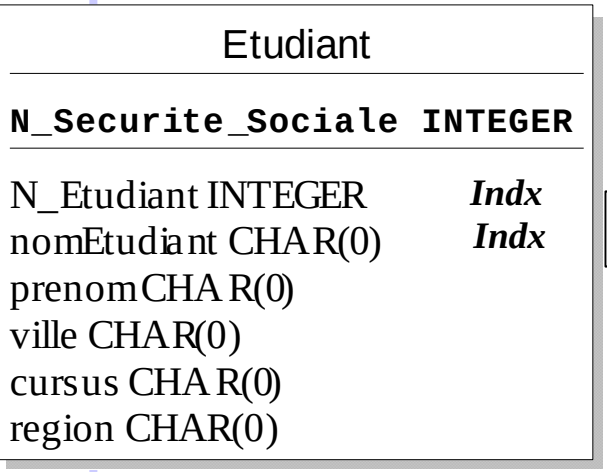



```
CREATE TABLE Etudiant
(

)
```

1. SQL, un langage pour décrire des données

- Expression des attributs d'une table relationnelle à l'aide de SQL



SQL

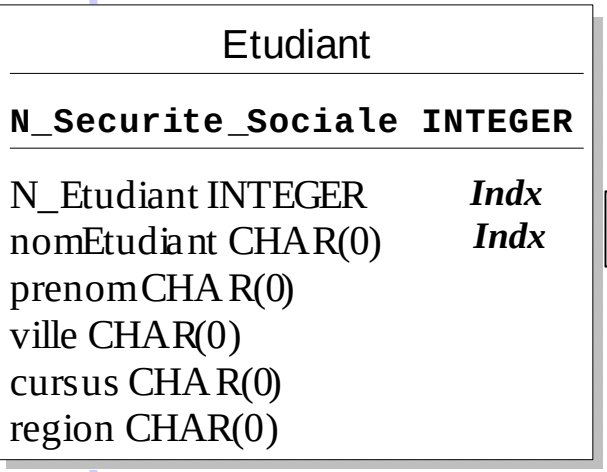
```
CREATE TABLE Etudiant
(
  N_Securite_Social NUMBER(0),
  N_Etudiant NUMBER(0),
  nomEtudiant CHAR(0),
  prenom CHAR(0),
  ville CHAR(0),
  region CHAR(0),
  cursus CHAR(0),




)
```

1. SQL, un langage pour décrire des données

- Expression d'une clé primaire d'une table relationnelle à l'aide de SQL



SQL

```
CREATE TABLE Etudiant
(
  N_Securite_Social NUMBER(0),
  N_Etudiant NUMBER(0),
  nomEtudiant CHAR(0),
  prenom CHAR(0),
  ville CHAR(0),
  region CHAR(0),
  cursus CHAR(0),

  CONSTRAINT PK_Etudiant PRIMARY KEY
  ( N_Securite_Social )
)
```

1. SQL, un langage pour décrire des données

- Expression d'un index d'une table relationnelle à l'aide de SQL

Etudiant	
N_Securite_Sociale INTEGER	
N_Etudiant INTEGER	***Indx***
nomEtudiant CHAR(0)	***Indx***
prenom CHAR(0)	
ville CHAR(0)	
cursus CHAR(0)	
region CHAR(0)	

SQL

```
CREATE TABLE Etudiant
(
  N_Securite_Social NUMBER(0),
  N_Etudiant NUMBER(0),
  nomEtudiant CHAR(0),
  prenom CHAR(0),
  ville CHAR(0),
  region CHAR(0),
  cursus CHAR(0),
  CONSTRAINT PK_Etudiant PRIMARY KEY
  ( N_Securite_Social )
)
```

```
ALTER TABLE Etudiant
ADD CONSTRAINT IDX_N_Etudiant UNIQUE
(
  N_Etudiant
);
CREATE UNIQUE INDEX IDX_N_Etudiant
ON Etudiant
(
  N_Etudiant
)
```

1. SQL, un langage pour décrire des données

- Expression d'un index d'une table relationnelle à l'aide de SQL

Etudiant	
N_Securite_Sociale INTEGER	
N_Etudiant INTEGER	*Indx*
nomEtudiant CHAR(0)	*Indx*
prenom CHAR(0)	
ville CHAR(0)	
cursus CHAR(0)	
region CHAR(0)	

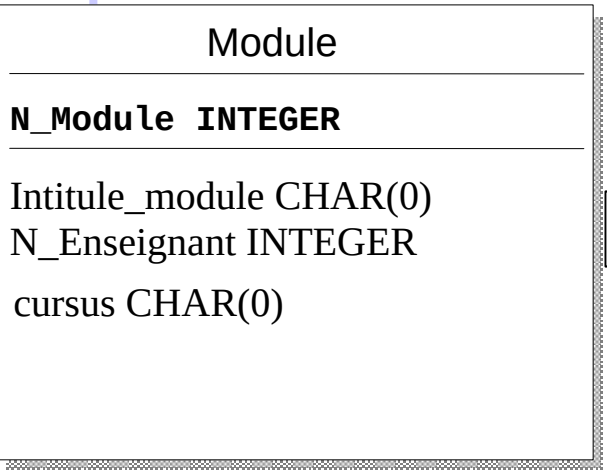
SQL

```
CREATE TABLE Etudiant
(
  N_Securite_Social NUMBER(0),
  N_Etudiant NUMBER(0),
  nomEtudiant CHAR(0),
  prenom CHAR(0),
  ville CHAR(0),
  region CHAR(0),
  cursus CHAR(0),
  CONSTRAINT PK_Etudiant PRIMARY KEY
  ( N_Securite_Social )
)
```

```
ALTER TABLE Etudiant
ADD CONSTRAINT IDX_nomEtudiant UNIQUE
(
  nomEtudiant
);
CREATE UNIQUE INDEX IDX_ nomEtudiant
ON Etudiant
(
 nomEtudiant
)
```

1. SQL, un langage pour décrire des données

- Expression d'une table relationnelle à l'aide de SQL

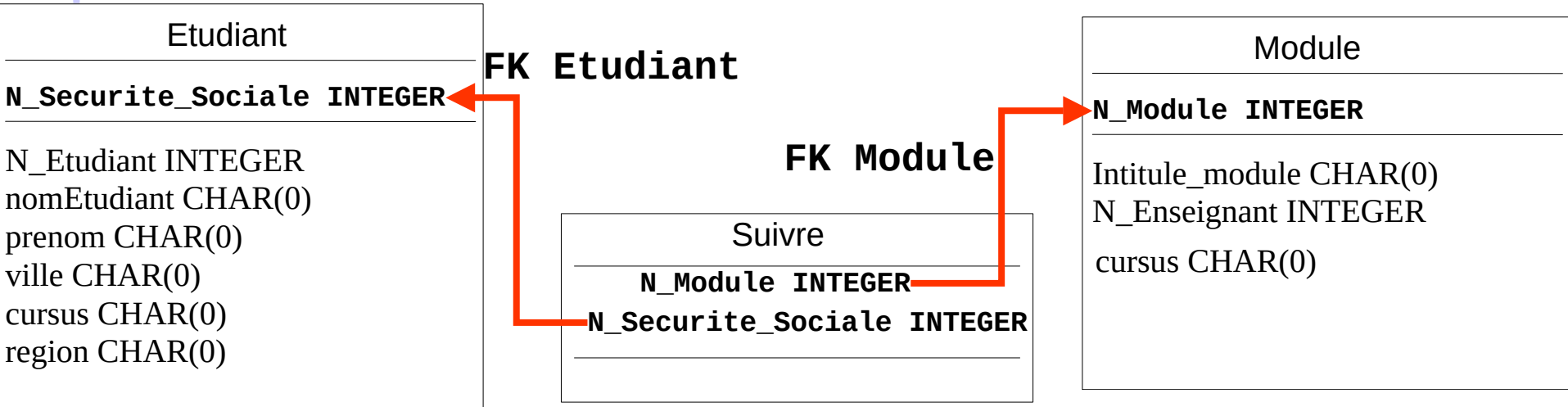


SQL

```
CREATE TABLE Module
(
  N_Module NUMBER(0) NOT NULL,
  Intitule_module CHAR(0) NOT NULL,
  N_Enseignant NUMBER(0) NOT NULL,
  cursus CHAR(0),
  CONSTRAINT PK_Module PRIMARY KEY
  (N_Module )
)
```

1. SQL, un langage pour décrire des données

- Expression d'une association entre deux tables relationnelles à l'aide de SQL




```
CREATE TABLE Suivre
(
  N_Module NUMBER(0),
  N_Securite_Sociale NUMBER(0)
)
```

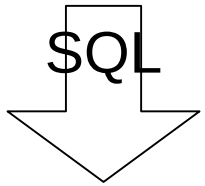
```
ALTER TABLE Suivre
ADD CONSTRAINT FK_Etudiant FOREIGN KEY
(N_Securite_Sociale
) REFERENCES Etudiant;

ALTER TABLE Suivre
ADD CONSTRAINT FK_Module FOREIGN KEY
(  N_Module
) REFERENCES Module;
```

2. SQL, un langage pour interroger une base de données

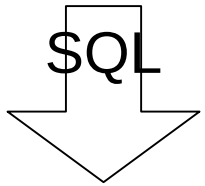
- Expression d'une projection π à l'aide de SQL

$\pi_{\text{Liste d'attributs}}$(Nom de la relation)



SELECT *liste d'attributs* FROM *Nom de la relation*

$\pi_{\text{nomEtudiant,cursus}}$ (Étudiants)

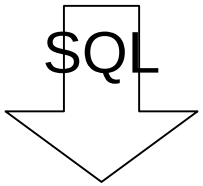


SELECT nomÉtudiant,cursus FROM Étudiants

2. SQL, un langage pour interroger une base de données

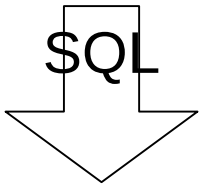
- Expression d'une sélection σ à l'aide de SQL

$\sigma_{\text{critère}}$ (Nom de la relation)



SELECT * FROM *Nom de la relation* WHERE *qualification*
(=,<,>,>=,<=, NOT LIKE)

$\sigma_{\text{régions = 'Lorraine'}}$ (Étudiants)

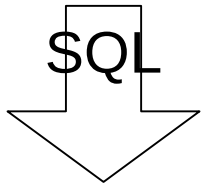


SELECT * FROM Étudiants WHERE régions=Lorraine

2. SQL, un langage pour interroger une base de données

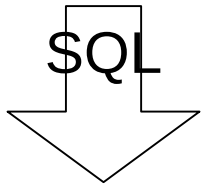
- Expression de la composition de la sélection σ et de la projection π à l'aide de SQL

$$\pi_{\text{Liste d'attributs}}(\sigma_{\text{critère}}(\text{Nom de la relation}))$$



SELECT *liste d'attributs* FROM *Nom de la relation* WHERE *qualification*
(=,<,>,>=,<=, NOT LIKE)

$$\pi_{\text{nomEtudiant,cursus}}(\sigma_{(10<=\text{Note}<=12)}(\text{Étudiants}))$$

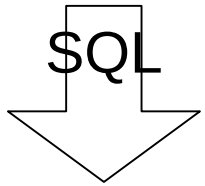


SELECT nomÉtudiant,cursus FROM Étudiants
WHERE Note>=10 AND Note<=12

2. SQL, un langage pour interroger une base de données

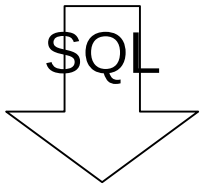
- Expression de la jointure θ à l'aide de SQL *1° formulation*

$\theta_{(critère)}$ (Liste de Noms de relation)



SELECT * FROM *Liste de Noms de relations* WHERE *expression de jointure*

$\theta_{(Étudiant.Cursus = Cours.Cursus)}$ (Étudiants,Cours)



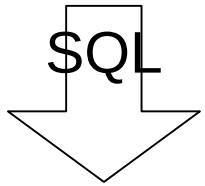
SELECT * FROM Étudiants , Cours

WHERE Étudiant.Cursus = Cours.Cursus

2. SQL, un langage pour interroger une base de données

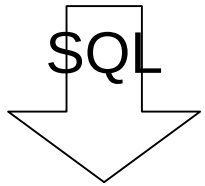
- Expression de la jointure θ à l'aide de SQL *2° formulation*

$\theta_{(critère)}$ (Liste de Noms de relation)



SELECT * FROM *Noms de relation 1* WHERE *expression de jointure* IN
(SELECT *expression de jointure* FROM *Noms de relation 2*

$\theta_{(Étudiant.Cursus = Cours.Cursus)}$ (Étudiants,Cours)

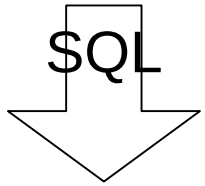


SELECT * FROM Cours
WHERE Cursus IN (SELECT Cursus FROM Étudiant)

2. SQL, un langage pour interroger une base de données

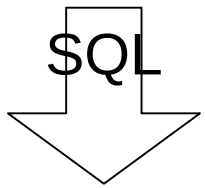
- Expression du produit cartésien χ à l'aide de SQL

Nom de la relation 1 χ Nom de la relation 2



SELECT * FROM *Liste de Noms de relations*

Étudiants χ Cours



SELECT * FROM Étudiants, Cours

2. SQL, un langage pour interroger une base de données

- Expression de valeur inconnue (NULL) à l'aide de SQL
  - Des instances d'attributs d'une relation peuvent ne pas être renseignés ou peuvent être sans sens pour l'enregistrement
  - Ces instances ne sont ni un zéro ni une chaîne vide
  - Cette absence de valeur est appelée valeur inconnue et notée NULL
  - Afin d'éviter qu'un attribut d'une relation soit de valeur inconnue, on lui impose une propriété NOT NULL

Table de vérité du ET et du OU étendue à des relations binaires

ET	*Vrai*	*Faux*	*Null*
Vrai	**Vrai**	**Faux**	**Null**
Faux	**Faux**	**Faux**	**Faux**
Null	**Null**	**Faux**	**Null**

OU	*Vrai*	*Faux*	*Null*
Vrai	**Vrai**	**Vrai**	**Vrai**
Faux	**Vrai**	**Faux**	**Null**
Null	**Vrai**	**Null**	**Null**

2. SQL, un langage pour interroger une base de données

- Expression d'une extension de la jointure : la jointure externe OUTER JOIN

  - La jointure externe étend le résultat de la jointure à des instances d'attributs d'une relation qui ne figurent pas dans l'autre relation mais dont on souhaite obtenir des informations. La jointure externe attribue la valeur NULL à ces instances.

SELECT *Liste des attributs* FROM *Liste de Noms de relations*
WHERE *Noms de relation 1.attribut* *= *Noms de relation 2.Attribut*

2. SQL, un langage pour interroger une base de données

- Expression d'un ordonnancement, d'une agrégation et d'un groupement de données

  - *Ordonnancement de données* ORDER BY

SELECT * FROM *Nom de relation* WHERE *qualification*
ORDER BY *Attribut ASC, Attribut DESC*

SELECT * FROM Étudiants WHERE Note>=10 AND Note<=12
ORDER BY nomÉtudiant ASC

2. SQL, un langage pour interroger une base de données

- Expression d'un ordonnancement, d'une agrégation et d'un groupement de données

  - *Agrégation de données*

    - COUNT ({*|{ [ALL | DISTINCT] } expression}) : cardinal de relation
    - SUM([ALL | DISTINCT] expression) : Somme d'une colonne
    - AVG([ALL | DISTINCT] expression) : Moyenne d'une colonne
    - MAX([ALL | DISTINCT] expression) : Maximum d'une colonne
    - MIN([ALL | DISTINCT] expression) : Minimum d'une colonne

2. SQL, un langage pour interroger une base de données

- Expression d'un ordonnancement, d'une agrégation et d'un groupement de données
  - *Agrégation de données* COUNT

SELECT COUNT (*) FROM *Nom de relation* WHERE *qualification*

SELECT COUNT (*) FROM Étudiants WHERE Note>=10 AND Note<=12

2. SQL, un langage pour interroger une base de données

- Expression d'un ordonnancement, d'une agrégation et d'un groupement de données
  - *Agrégation de données* SUM

SELECT SUM (*Attribut)* FROM *Nom de relation* WHERE *qualification*
SELECT *Attribut* FROM *Nom de relation* WHERE SUM et *qualification*

SELECT NomÉtudiant FROM BonsPoints WHERE SUM (Points)>=100

2. SQL, un langage pour interroger une base de données

- Expression d'un ordonnancement, d'une agrégation et d'un groupement de données
  - *Agrégation de données* AVG

SELECT AVG (*Attribut)* FROM *Nom de relation* WHERE *qualification*
SELECT *Attribut* FROM *Nom de relation* WHERE AVG et *qualification*

SELECT NomÉtudiant FROM Étudiants WHERE AVG (Note)>=10

2. SQL, un langage pour interroger une base de données

- Expression d'un ordonnancement, d'une agrégation et d'un groupement de données
  - *Agrégation de données* MAX, MIN

SELECT MAX,MIN (*Attribut)* FROM *Nom de relation* WHERE *qualification*
SELECT *Attribut* FROM *Nom de relation* WHERE MAX,MIN et *qualification*

SELECT NomÉtudiant FROM Étudiants

WHERE Note IN (SELECT MAX(Note) FROM Étudiant)

2. SQL, un langage pour interroger une base de données

- Expression d'un ordonnancement, d'une agrégation et d'un groupement de données
  - *Groupement de données*

SELECT *Attribut* FROM *Nom de relation* GROUP BY *Attribut* HAVING *critère*

SELECT COUNT(NomÉtudiant) FROM Étudiants GROUP BY Cursus

SELECT NombreÉtudiants FROM Promotion GROUP BY Cursus

HAVING NombreÉtudiants<10

3. SQL, un langage pour mettre à jour une base de données

- Expression d'un ajout de données INSERT INTO

INSERT INTO *Nom de relation (Liste des attributs)* {VALUES (expression-constante1,…, expression-constanteN)}

INSERT INTO Étudiants(NomÉtudiant,prénom,ville,région,cursus)
VALUES (Bayard,Simon,Annecy,Savoie,2°AI)

3. SQL, un langage pour mettre à jour une base de données

- Expression d'une modification de données UPDATE

UPDATE *Nom de relation* SET *Attributs*=expression
[FROM …] [WHERE *qualification*]

UPDATE Étudiants SET Note =0 WHERE Cursus=1°AI

3. SQL, un langage pour mettre à jour une base de données

- Expression d'une modification de données DELETE

DELETE [FROM] *Liste des Noms de relation* [WHERE *qualification*]

DELETE Étudiants WHERE Note <=10

4. Récapitulatif

Pour décrire des données (créer une base de données)

- Créer une table de données : CREATE TABLE A (ATTRIBUT, PK, INDEX)

- Créer une relation entre tables de données :
  - ALTER TABLE A
    ADD CONSTRAINT FK_nom FOREIGN KEY

  - ALTER TABLE B
    ADD CONSTRAINT FK_nom FOREIGN KEY

  - CREATE TABLE AB ALTER TABLE AB
    ADD CONSTRAINT FK_nom FOREIGN KEY

4. Récapitulatif

Pour manipuler des instances de données

- créer des instances de données
  - INSERT INTO Nom de relation (Liste des attributs) {VALUES (expression-constante1,…, expression-constanteN)}
- Pour supprimer des instances de données
  - DELETE [FROM] Liste des Noms de relation [WHERE qualification]
- Pour mettre à jour des instances de données
  - UPDATE Nom de relation SET Attributs=expression [FROM …] [WHERE qualification]

4. Récapitulatif

- Pour interroger des instances de données d'une table

  - Consulter des instances de données

    - SELECT * FROM Nom de la relation WHERE qualification

  - Pour raffiner la consultation des instances de données

    - SELECT liste d'attributs FROM Nom de la relation
    - SELECT * FROM Liste de Noms de relations WHERE expression de jointure
    - SELECT Liste des attributs FROM Liste de Noms de relations WHERE Noms de relation 1.attribut *= Noms de relation 2.Attribut

  - Pour combiner des instances de données

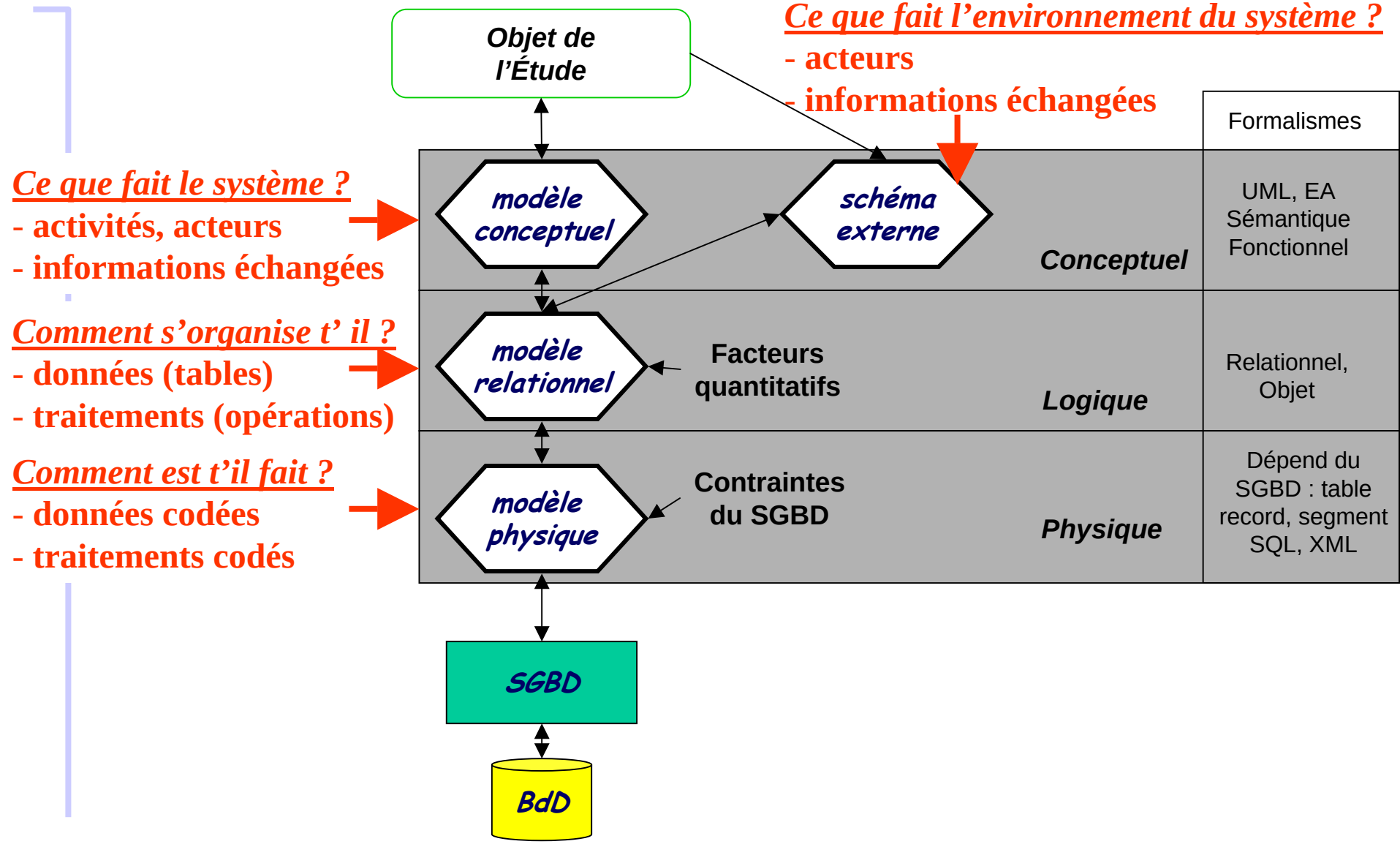
    - SELECT * FROM Liste de Noms de relations

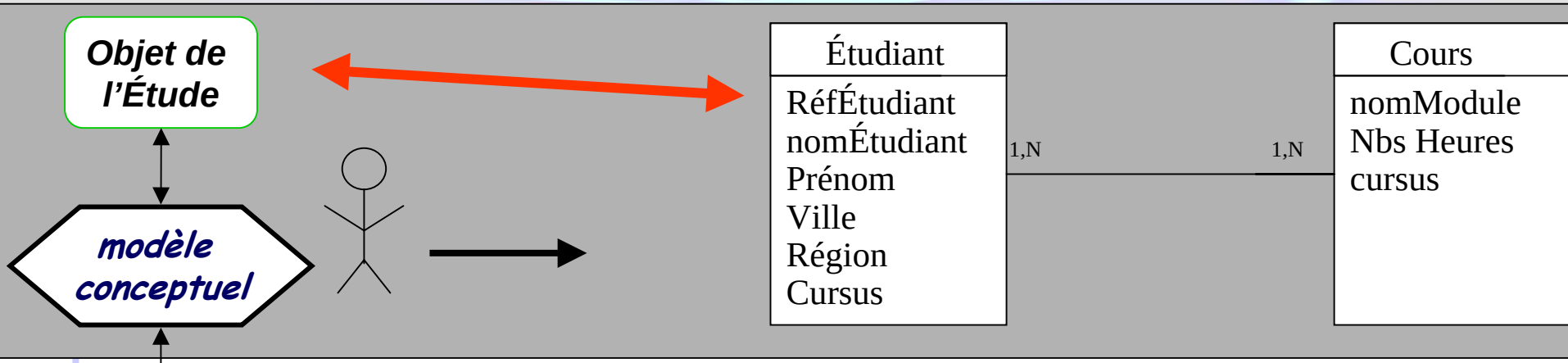
4. Récapitulatif

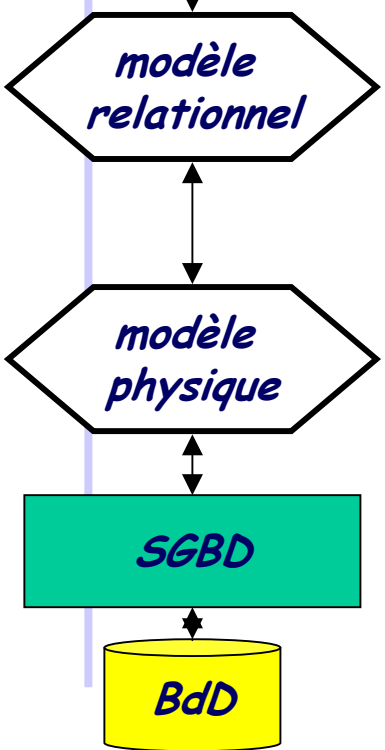
Pour créer des opérations d'instances de données

- Pour ordonner des instances de données
  - SELECT * FROM Nom de relation WHERE qualification ORDER BY Attribut ASC, Attribut DESC
- Pour agréger des instances de données
  - COUNT ({*|{ [ALL | DISTINCT] } expression}) : cardinal de relation
  - SUM([ALL | DISTINCT] expression) : Somme d'une colonne
  - AVG([ALL | DISTINCT] expression) : Moyenne d'une colonne
  - MAX([ALL | DISTINCT] expression) : Maximum d'une colonne
  - MIN([ALL | DISTINCT] expression) : Minimum d'une colonne
- Pour grouper des instances de données
  - SELECT Attribut FROM Nom de relation GROUP BY Attribut HAVING critère

4. Récapitulatif







nomÉtudiant	prénom	ville	région	cursus

1 ∞

$\sigma_{\text{régions = 'Lorraine'}}$ (Étudiants)

nomModule	Nbs Heures	cursus

```
CREATE TABLE Étudiant (RéfÉtudiant INT NOT NULL; nomÉtudiant VAR CHAR (20);
Prénom  VAR CHAR (20); Ville VAR CHAR (20); Région VAR CHAR (20); Cursus  INT NOT NULL )
SELECT [nomÉtudiant], [Prénom], [Ville], [Région], [Cursus] FROM Étudiants
WHERE ((([Étudiants].[Région])="Lorraine"));
```